1. Historias de Usuario.

├── apps
│ ├── atracciones
│ │ ├── __init__.py

```
from flask import Blueprint

blueprint = Blueprint(
    'atracciones_blueprint',
    __name__,
    url_prefix=''
)
```

│ │ ├── models.py

```
from apps import db


class Atracciones(db.Model):

    __tablename__ = 'Atracciones'

    id_atraccion = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(255), nullable=False)
    descripcion = db.Column(db.Text)
    id_pais = db.Column(db.Integer, nullable=False)
    url_foto = db.Column(db.Text)
    estado = db.Column(db.String(50))
    maps = db.Column(db.Text)

    __table_args__ = (
        db.PrimaryKeyConstraint('id_atraccion', 'id_pais', name='PK_ATRACCIONES'),
    )

    def __init__(self, id_pais: int, nombre: str, descripcion: str, url_foto: str,
estado: str, maps: str):
        self.id_pais = id_pais
        self.nombre = nombre
        self.descripcion = descripcion
        self.maps = maps
        self.url_foto = url_foto
        self.estado = estado

    def __repr__(self):
        return f"<Atracciones(id_atraccion={self.id_atraccion}, id_pais={self.id_pais},
nombre={self.nombre}, descripcion={self.descripcion}, url_foto={self.url_foto},
estado={self.estado})>"









    def to_string (self) -> dict:
```

```
        """Returns a dictionary with the details of the attraction."""
        return {
            'id_atraccion': self.id_atraccion,
            'id_pais': self.id_pais,
            'nombre': self.nombre,
            'descripcion': self.descripcion,
            'url_foto': self.url_foto,
            'estado': self.estado
        }

    def save(self):
        db.session.add(self)
        db.session.commit()

    @staticmethod
    def get_all():
        return Atracciones.query.all()

    @staticmethod
    def find_by_id(id_atraccion):
        return Atracciones.query.get(id_atraccion)

    @staticmethod
    def find_by_user(id_usuario):
        return Atracciones.query.filter_by(id_usuario=id_usuario).all()

    @staticmethod
    def update(self, new_data):
        for key, item in new_data.items():
            setattr(self, key, item)
        db.session.commit()

    @staticmethod
    def where(**kwargs):
        return Atracciones.query.filter_by(**kwargs).all()

    def delete(self):
        db.session.delete(self)
        db.session.commit()
```

```
        │   ├── __pycache__
        │   └── routes.py
```

```
from flask import render_template, redirect, request, url_for, Blueprint
from flask_login import (
    current_user,
    login_user,
    logout_user, login_required
)

from apps import login_manager
from apps.atracciones import blueprint
from apps.atracciones.models import Atracciones


blueprint = Blueprint('atracciones_blueprint', __name__, url_prefix='/atracciones',
static_folder='../static')




@blueprint.route('/index/<int:idPais>')
```

```
@login_required
def index(idPais) -> 'html':
    lista_atracciones = Atracciones.where(id_pais=idPais)
    return render_template('home/index.html', segment='index',
atracciones=lista_atracciones
                           , url_atracciones='/turistic-place',
urlA='/atracciones/index/')

@login_manager.unauthorized_handler
def unauthorized_handler():
    return render_template('home/page-403.html'), 403


@blueprint.errorhandler(403)
def access_forbidden(error):
    return render_template('home/page-403.html'), 403


@blueprint.errorhandler(404)
def not_found_error(error):
    return render_template('home/page-404.html'), 404


@blueprint.errorhandler(500)
def internal_error(error):
    return render_template('home/page-500.html'), 500
```

```
|   ├── authentication
|   |   ├── forms.py
```

```
from flask_wtf import FlaskForm
from wtforms import TextField, PasswordField
from wtforms.validators import Email, DataRequired

# login and registration


class LoginForm(FlaskForm):
    username = TextField('Username',
                         id='username_login',
                         validators=[DataRequired()])
    password = PasswordField('Password',
                             id='pwd_login',
                             validators=[DataRequired()])


class CreateAccountForm(FlaskForm):
    username = TextField('Username',
                         id='username_create',
                         validators=[DataRequired()])
    email = TextField('Email',
                      id='email_create',
                      validators=[DataRequired(), Email()])
    password = PasswordField('Password',
                             id='pwd_create',
                             validators=[DataRequired()])
```

│ │ ├── __init__.py

```
# -*- encoding: utf-8 -*-
"""
Copyright (c) 2023 - present deplooymentsoftware@gmail.com
"""

from flask import Blueprint

blueprint = Blueprint(
    'authentication_blueprint',
    __name__,
    url_prefix=''
)
```

│ │ ├── models.py

```
from flask_login import UserMixin

from apps import db, login_manager

from apps.authentication.util import hash_pass

class Users(db.Model, UserMixin):

    __tablename__ = 'Users'

    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True)
    email = db.Column(db.String(64), unique=True)
    password = db.Column(db.LargeBinary)

    def __init__(self, **kwargs):
        for property, value in kwargs.items():
            # depending on whether value is an iterable or not, we must
            # unpack it's value (when **kwargs is request.form, some values
            # will be a 1-element list)
            if hasattr(value, '__iter__') and not isinstance(value, str):
                # the ,= unpack of a singleton fails PEP8 (travis flake8 test)
                value = value[0]

            if property == 'password':
                value = hash_pass(value)  # we need bytes here (not plain str)

            setattr(self, property, value)

    def __repr__(self):
        return str(self.username)

@login_manager.user_loader
def user_loader(id):
    return Users.query.filter_by(id=id).first()

@login_manager.request_loader
def request_loader(request):
    username = request.form.get('username')
    user = Users.query.filter_by(username=username).first()
    return user if user else None
```

```
def save(self):
    db.session.add(self)
    db.session.commit()

@staticmethod
def get_all():
    return Users.query.all()

@staticmethod
def find_by_user(id_usuario):
    return Users.query.filter_by(id_usuario=id_usuario).all()

@staticmethod
def update(self, new_data):
    for key, item in new_data.items():
        setattr(self, key, item)
    db.session.commit()

@staticmethod
def where(**kwargs):
    return Users.query.filter_by(**kwargs).all()

def delete(self):
    db.session.delete(self)
    db.session.commit()
```

```
        │   ├── __pycache__
        │   ├── routes.py
```

```
from flask import render_template, redirect, request, url_for
from flask_login import (
    current_user,
    login_user,
    logout_user
)

from apps import db, login_manager
from apps.authentication import blueprint
from apps.authentication.forms import LoginForm, CreateAccountForm
from apps.authentication.models import Users

from apps.authentication.util import verify_pass


@blueprint.route('/')
def route_default():
    return redirect(url_for('authentication_blueprint.login'))


# Login & Registration

@blueprint.route('/login', methods=['GET', 'POST'])
def login():
    login_form = LoginForm(request.form)
    if 'login' in request.form:

        # read form data
        username = request.form['username']
        password = request.form['password']


        # Locate user
```

```
        user = Users.query.filter_by(username=username).first()

        # Check the password
        if user and verify_pass(password, user.password):

            login_user(user)
            return redirect(url_for('authentication_blueprint.route_default'))

        # Something (user or pass) is not ok
        return render_template('accounts/login.html',
                               msg='Wrong user or password',
                               form=login_form)

    if not current_user.is_authenticated:
        return render_template('accounts/login.html',
                               form=login_form)
    return redirect(url_for('home_blueprint.index'))


@blueprint.route('/register', methods=['GET', 'POST'])
def register():
    create_account_form = CreateAccountForm(request.form)
    if 'register' in request.form:

        username = request.form['username']
        email = request.form['email']

        # Check usename exists
        user = Users.query.filter_by(username=username).first()
        if user:
            return render_template('accounts/register.html',
                                   msg='Username already registered',
                                   success=False,
                                   form=create_account_form)

        # Check email exists
        user = Users.query.filter_by(email=email).first()
        if user:
            return render_template('accounts/register.html',
                                   msg='Email already registered',
                                   success=False,
                                   form=create_account_form)

        # else we can create the user
        user = Users(**request.form)
        db.session.add(user)
        db.session.commit()

        return render_template('accounts/register.html',
                               msg='User created please <a href="/login">login</a>',
                               success=True,
                               form=create_account_form)

    else:
        return render_template('accounts/register.html', form=create_account_form)


@blueprint.route('/logout')
def logout():
    logout_user()
    return redirect(url_for('authentication_blueprint.login'))

# Errors
```

```
@login_manager.unauthorized_handler
def unauthorized_handler():
    return render_template('home/page-403.html'), 403


@blueprint.errorhandler(403)
def access_forbidden(error):
    return render_template('home/page-403.html'), 403


@blueprint.errorhandler(404)
def not_found_error(error):
    return render_template('home/page-404.html'), 404


@blueprint.errorhandler(500)
def internal_error(error):
    return render_template('home/page-500.html'), 500
```

```
        │   │   ├── src
        │   │   │   ├── __init__.py
```

```
from flask import Blueprint

blueprint = Blueprint(
    'authentication_src_blueprint',
    __name__,
    url_prefix=''
)
```

```
        │   │   │   └── Login
                ├── application
                │   ├── commands.py
```

```
from abc import ABC, abstractmethod

class Command(ABC):
    """Base Command class"""

class CreateUserCommand(Command):
    def __init__(self, username: str, email: str, password: str):
        self.username = username
        self.email = email
        self.password = password

class CommandHandler(ABC):
    @abstractmethod
    def handle(self, command: Command):
        pass

class CreateUserCommandHandler(CommandHandler):
    def __init__(self, user_write_repository):
        self.user_write_repository = user_write_repository

    def handle(self, command: CreateUserCommand):
        user = UserWrite(username=command.username, email=command.email,
password=command.password)
        self.user_write_repository.create(user)
```

├── __init__.py

```
from flask import Blueprint

blueprint = Blueprint(
    'authentication_src_login_application_blueprint',
    __name__,
    url_prefix=''
)
```

│   ├── queries.py

```
from abc import ABC, abstractmethod

from apps.authentication.src.Login.domain.interfaces.userResponse import UserResponse


class Query(ABC):
    """Base Query class"""

class GetAllUsersQuery(Query):
    pass

class GetUserByIdQuery(Query):
    def __init__(self, user_id: int):
        self.user_id = user_id

class QueryHandler(ABC):
    @abstractmethod
    def handle(self, query: Query):
        pass

class GetAllUsersQueryHandler(QueryHandler):
    def __init__(self, user_read_repository):
        self.user_read_repository = user_read_repository

    def handle(self, query: GetAllUsersQuery):
        users = self.user_read_repository.get_all()
        return [UserResponse.from_entity(user) for user in users]

class GetUserByIdQueryHandler(QueryHandler):
    def __init__(self, user_read_repository):
        self.user_read_repository = user_read_repository

    def handle(self, query: GetUserByIdQuery):
        user = self.user_read_repository.get_by_id(query.user_id)
        return UserResponse.from_entity(user) if user else None
```

│   └── services.py

├── domain

│   ├── entities

│   │   ├── __init__.py

```
from flask import Blueprint

blueprint = Blueprint(
    'authentication_src_login_domain_blueprint',
    __name__,
    url_prefix=''
)
```

│ │ ├── UserRead.py

```
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from apps import db

Base = declarative_base()

class UserRead(db.Model):
    __tablename__ = 'Users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True, nullable=False)
    email = db.Column(db.String(64), unique=True, nullable=False)
    password = db.Column(db.LargeBinary, nullable=False)

    def __init__(self, username, email, password):
        self.username = username
        self.email = email
        self.password = password

    def __repr__(self):
        return f"<User(id={self.id}, username={self.username}, email={self.email})>"
```

│ │ └── UserWrite.py

```
from sqlalchemy import create_engine, Column, Integer, String, Sequence
from sqlalchemy.ext.declarative import declarative_base
from apps import db

Base = declarative_base()

class UserWrite(db.Model):
    __tablename__ = 'Users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True)
    email = db.Column(db.String(64), unique=True)
    password = db.Column(db.LargeBinary)
```

│ ├── __init__.py

```
from flask import Blueprint

blueprint = Blueprint(
    'authentication_src_blueprint',
    __name__,
    url_prefix=''
)
```

│ └── interfaces

│ ├── __init__.py

```
from flask import Blueprint

blueprint = Blueprint(
    'authentication_src_login_domain_interfaces_blueprint',
    __name__,
    url_prefix=''
)
```

│ ├── userRepositoryRead.py

```python
from abc import ABC, abstractmethod


class UserReadRepository(ABC):

    @abstractmethod
    def get_all(self):
        """Método para obtener todos los usuarios."""
        pass

    @abstractmethod
    def get_by_id(self, user_id: int):
        """Método para obtener un usuario por su ID."""
        pass

    @abstractmethod
    def get_by_custom_function(self, param):
        """Método para ejecutar una función personalizada y obtener resultados."""
        pass

    @abstractmethod
    def call_stored_procedure(self, param1, param2):
        """Método para ejecutar un procedimiento almacenado."""
        pass
```

│ ├── userRepositoryWrite.py

```python
# userRepositoryWrite.py
from abc import ABC, abstractmethod

class UserWriteRepository(ABC):
    @abstractmethod
    def create(self, user):
        pass

    @abstractmethod
    def update(self, user):
        pass

    @abstractmethod
    def delete(self, user_id):
        pass
```

│ └── userResponse.py

```python
from apps.authentication.src.Login.domain.entities.UserRead import UserRead


class UserResponse:
    def __init__(self, id, username, email):
        self.id = id
        self.username = username
        self.email = email

    @staticmethod
    def from_entity(user: UserRead) -> 'UserResponse':
        return UserResponse(id=user.id, username=user.username, email=user.email)
class UsersResponses:
    def __init__(self, users):
        self.users = [UserResponse(id=user.id, username=user.username, email=user.email)
for user in users]
```

├── infrastructure
│   ├── __init__.py

```
from flask import Blueprint

blueprint = Blueprint(
    'authentication_src_login_infrastructure_blueprint',
    __name__,
    url_prefix=''
)
```

│   ├── postgresUserReadRepository.py

```
from apps import db
from sqlalchemy import create_engine, text
from apps.authentication.src.Login.domain.interfaces.userRepositoryRead import
UserReadRepository
from apps.authentication.src.Login.domain.entities.UserRead import UserRead

session = db.session

class UserReadRepositoryPostgres(UserReadRepository):
    def get_all(self):
        with session.begin(subtransactions=True):
            return session.query(UserRead).all()

    def get_by_id(self, user_id):
        with session.begin(subtransactions=True):
            return session.query(UserRead).filter_by(id=user_id).first()

    def get_by_custom_function(self, param):
        with session.begin(subtransactions=True):
            sql = text("SELECT * FROM my_function(:param)")
            result = session.execute(sql, {'param': param}).fetchall()
            # Aquí podrías mapear 'result' a tus entidades o devolverlo como está
            return result

    def call_stored_procedure(self, param1, param2):
        with session.begin(subtransactions=True):
            sql = text("CALL my_procedure(:param1, :param2)")
            session.execute(sql, {'param1': param1, 'param2': param2})
```

│   └── postgresUserWriteRepository.py

```
# UserWriteRepositoryPostgres.py
from sqlalchemy import create_engine, text
from apps import db
from apps.authentication.src.Login.domain.interfaces.userRepositoryWrite import
UserWriteRepository
from apps.authentication.src.Login.domain.entities.UserWrite import UserWrite

session = db.session

class UserWriteRepositoryPostgres(UserWriteRepository):
    def create(self, user):
        with session.begin():
            session.add(user)

    def update(self, user):
        with session.begin():
            session.merge(user)

    def delete(self, user_id):
```

```
        with session.begin():
            user = session.query(UserWrite).filter_by(id=user_id).first()
            if user:
                session.delete(user)
```

└── __init__.py

```
from flask import Blueprint

blueprint = Blueprint(
    'authentication_src_login_blueprint',
    __name__,
    url_prefix=''
)
```

│ │ └── util.py

```
# -*- encoding: utf-8 -*-
"""
Copyright (c) 2023 - present deplooymentsoftware@gmail.com
"""

import os
import hashlib
import binascii

# Inspiration -> https://www.vitoshacademy.com/hashing-passwords-in-python/


def hash_pass(password):
    """Hash a password for storing."""

    salt = hashlib.sha256(os.urandom(60)).hexdigest().encode('ascii')
    pwdhash = hashlib.pbkdf2_hmac('sha512', password.encode('utf-8'),
                                  salt, 100000)
    pwdhash = binascii.hexlify(pwdhash)
    return (salt + pwdhash)  # return bytes


def verify_pass(provided_password, stored_password):
    """Verify a stored password against one provided by user"""

    stored_password = stored_password.decode('ascii')
    salt = stored_password[:64]
    stored_password = stored_password[64:]
    pwdhash = hashlib.pbkdf2_hmac('sha512',
                                  provided_password.encode('utf-8'),
                                  salt.encode('ascii'),
                                  100000)
    pwdhash = binascii.hexlify(pwdhash).decode('ascii')
    return pwdhash == stored_password
```

│ ├── comentarios
│ │ ├── __init__.py
│ │ └── models.py
│ ├── config.py
│ ├── db.sqlite3

```
│   ├── favoritos
│   │   ├── __init__.py
│   │   └── models.py
│   ├── home
│   │   ├── __init_.py
│   │   ├── __pycache__
│   │   │   ├── __init__.cpython-311.pyc
│   │   │   └── routes.cpython-311.pyc
│   │   └── routes.py
│   ├── __init__.py
│   ├── paises
│   │   ├── __init__.py
│   │   ├── models.py
│   │   └── __pycache__
│   │       └── __init__.cpython-311.pyc
│   ├── __pycache__
│   │   ├── config.cpython-311.pyc
│   │   └── __init__.cpython-311.pyc
│   ├── static
│   │   └── assets
│   │       ├── css
│   │       ├── fonts
│   │       ├── gulpfile.js
│   │       ├── img
│   │       ├── js
│   │       ├── package.json
│   │       ├── scss
│   │       └── vendor
│   └── templates
│       ├── accounts
│       │   ├── login.html
│       │   └── register.html
│       ├── home
│       │   ├── descripcionpais.html
```

```
│   │   ├── faqs.html
│   │   ├── Feedback.html
│   │   ├── icons.html
│   │   ├── index.html
│   │   ├── login.html
│   │   ├── map.html
│   │   ├── page-403.html
│   │   ├── page-404.html
│   │   ├── page-500.html
│   │   ├── profile.html
│   │   ├── register.html
│   │   ├── tables.html
│   │   └── turistic-place.html
│   ├── includes
│   │   ├── footer-fullscreen.html
│   │   ├── footer.html
│   │   ├── navigation-fullscreen.html
│   │   ├── navigation.html
│   │   ├── scripts.html
│   │   └── sidenav.html
│   └── layouts
│       ├── base-fullscreen.html
│       └── base.html
├── CHANGELOG.md
├── docker-compose.yml
├── Dockerfile
├── gunicorn-cfg.py
├── LICENSE.md
├── media
│   ├── argon-dashboard-flask-intro.gif
│   ├── argon-dashboard-flask-screen-icons.png
│   ├── argon-dashboard-flask-screen-login.png
│   ├── argon-dashboard-flask-screen.png
│   ├── argon-dashboard-flask-screen-register.png
```

```
│   ├── argon-dashboard-flask-screen-tables.png
│   └── argon-dashboard-flask-screen-widgets.png
├── nginx
│   └── appseed-app.conf
├── package.json
├── Procfile
├── __pycache__
│   └── run.cpython-311.pyc
├── README.md
├── requirements-mysql.txt
├── requirements-pgsql.txt
├── requirements.txt
├── run.py
├── runtime.txt
└── venv
```